

REPAST FLOWCHART GETTING STARTED

ERIC TATARA & MICHAEL NORTH - REPAST DEVELOPMENT TEAM

0. BEFORE WE GET STARTED

Before we can do anything with Repast Symphony, we need to make sure that we have a proper installation of the latest version. Instructions on downloading and installing Repast Symphony on various platforms can be found on the [Repast website](#).¹ Repast Symphony requires Java 8 to be installed. Java can be found at the [Java Standard Edition Downloads Page](#).²

1. GETTING STARTED WITH REPAST FLOWCHART

We will be building a simple agent-based model involving zombies chasing humans and humans running away from zombies. When we are finished, the running model should look like fig. 1.

The first thing we must do is create a new Repast Symphony project. Assuming you've started Repast Symphony, right click in the Package Explorer pane and choose "New" then "Other". A dialog box comes up that allows you to select a wizard. Select "Repast Symphony Project" under the "Repast Symphony" folder (Fig. 2) and click "Next". This brings up the New Repast Symphony Project Wizard which gives us the ability to name our project (and a few more options which we'll ignore for now). Type "FlowZombies" in the "Project name" field, and press the "Finish" button.

By default Repast Symphony will hide many of the details of the project. This is appropriate for ReLogo projects but not for those written with Flowcharts or in Java. If the ReLogo filters have not been previously disabled then we need to do that now. If you click on the "+" (Windows) or triangle (OSX) next to "FlowZombies" and you see a variety of folders (batch, docs, etc), then the filter has been disabled. If you only see the src directory then the filter needs to be disabled. To disable the filter, click the downward pointing arrow in the "Package Explorer" pane. If you see "ReLogo Resource Filter" then click on it to disable the filter (Fig. 3). Otherwise, click on the "Filters" item. This brings up the Java Element Filters windows. Scroll through the elements and click off the checkbox for ReLogo Resource Filter (Fig. 4).

In addition, by default autobuilding is disabled. However, this is not appropriate for projects written with Flowcharts and in Java. To enable auto building go to the Project menu and select Build Automatically.

¹ <https://repast.github.io/download.html>

² <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

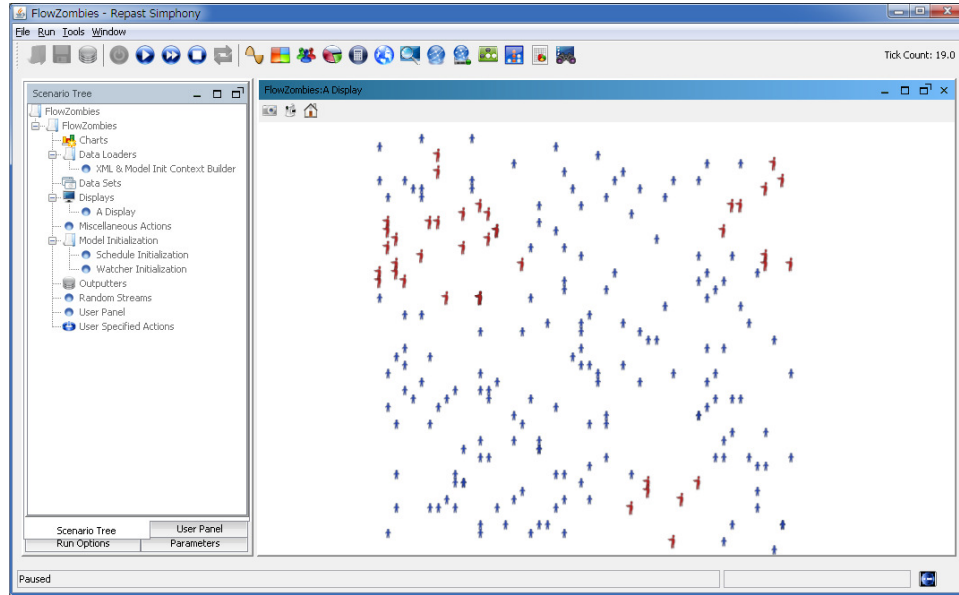


FIGURE 1. Completed Zombies Model

Lastly, if Eclipse has defaulted to the ReLogo perspective switch it to the Java one. The perspective selections can be found in the upper right hand corner (Fig. 5). Click the Java perspective.

1.1. Building the Model. Repast agents may be created through several different paths, such as pure Java agents, Groovy agents, or Flowchart agents. This tutorial will demonstrate agent design using the Flowchart method. The Repast Symphony project wizard creates a source directory and default package into which we can create these agent classes. In our case, the package is “FlowZombies”, which can be seen immediately under the src directory.³

To create a new Flowchart agent, right click on the created “FlowZombies” package located in the newly create project. Select “New”, then “Other” and browse to the “Repast Symphony” category (Fig. 6). Select “Agent” and in the File name, rename the default “Untitled.agent” to “Human.agent” (Fig. 7). Repeat this process for the Zombie agent, using the name “Zombie.agent”.

The `Zombie.agent` and `Human.agent` files (along with the automatically generated `Zombie.groovy` and `Human.groovy` files) should be visible in the FlowZombies packages underneath the src folder, and the Agent flowcharts should be open in Eclipse’s editor pane.

³Package names typically use an internet domain name as the basis for a package name, but for the purposes of this tutorial “FlowZombies” is perfectly fine. See [Java Tutorial: packages](#) for more info.

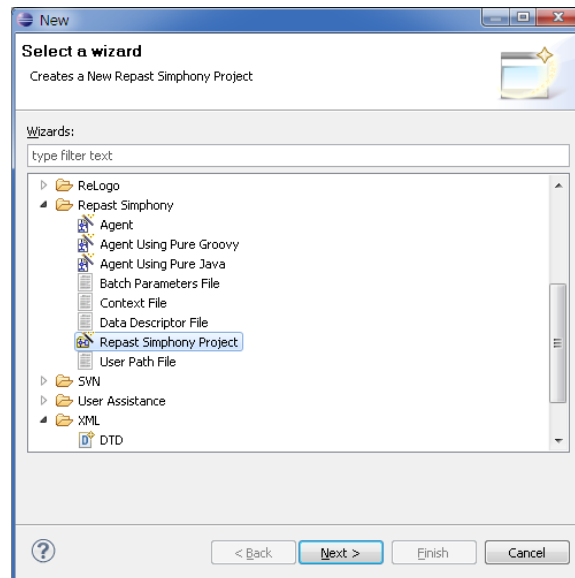


FIGURE 2. Repast Simphony Project Wizard.

(If they are not open, double click on each `Human.agent` file to open it). Let's begin with the Zombies. The Zombies behavior is to wander around looking for Human's to infect. More specifically, each iteration of the simulation, each Zombie will determine where the most Humans are within its local area and move there. Once there it will attempt to infect a Human at that location and turn it into a Zombie.

We will begin by implementing the movement. To do this we will locate the Zombies and Humans within a `Grid`. The `Grid` allows us to do neighborhood and proximity queries (i.e. "who is near me?") using discrete integer Grid coordinates.⁴ Let's begin by adding a property to the Zombie Agent that indicates whether the Zombie has moved. To create a Property block in the Flowchart, click on the Property icon the the Steps pane and then click on the Flowchart canvas. The Property block will appear and may be selected. Selecting an element in a flowchart allows the element to be edited using the Properties pane at the bottom of the window. If the Properties pane is not visible, right click on a flowchart element and select "Show Properties."

Select the Moved Property and in the Properties pane, enter the following information:

- (1) Step 2: Moved
- (2) Step 3: moved
- (3) Step 5: boolean
- (4) Step 6: false

⁴More details on `Grid` can found in the Repast Java API documentation and the Repast Reference.

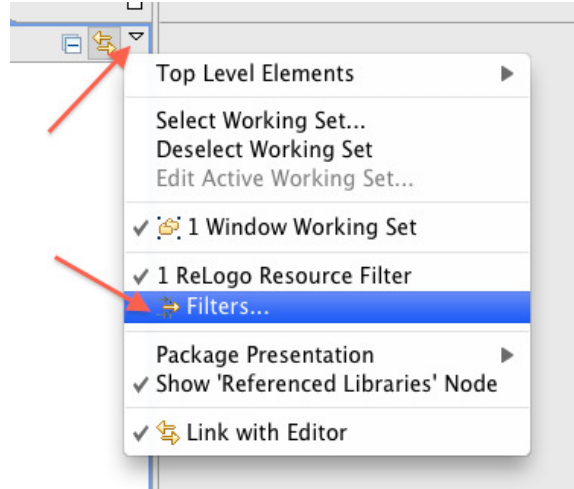


FIGURE 3. Disabling the ReLogo Filter.

This initial flowchart should look like Fig. 8. Now let's add a Behavior to Zombie that will be executed every iteration of the simulation. Select a new Behavior block from the Steps pane in the flowchart editor and add it to the flowchart (Fig. 9). In the properties pane, edit the following items for this Behavior:

- (1) Step 3a: Type in a Constant Starting Time for the Behavior : 1
- (2) Step 3c: Type in a Constant Repeat Interval for the Behavior: 1

Next, we will provide details for all elements of the movement Behavior. The completed flowchart for this behavior is shown in figure (Fig. 10). Add the various blocks from the Steps pane so that it looks like the completed flowchart in Fig. 10. Connections between the step blocks can be created by choosing the "Connection" option in the flow chart editor, clicking on a step block and dragging to the other. Once the flowchart steps and connections have been added, we will now proceed to edit each of the flowchart elements.

Task Grid Neighbors

- (1) `Grid grid = FindGrid("FlowZombies/grid")`
- (2) `GridPoint pt = grid.getLocation(this)`
- (3) `GridCellNgh nghCreator = new GridCellNgh(grid, pt, Human.class, 1, 1)`
- (4) `List gridCells = nghCreator.getNeighborhood(true)`
- (5) `SimUtilities.shuffle(gridCells, RandomHelper.getUniform())`

Task Initialize loop

- (1) `GridPoint pointWithMostHumans = null`
- (2) `int maxCount = -1`

Loop over neighbors

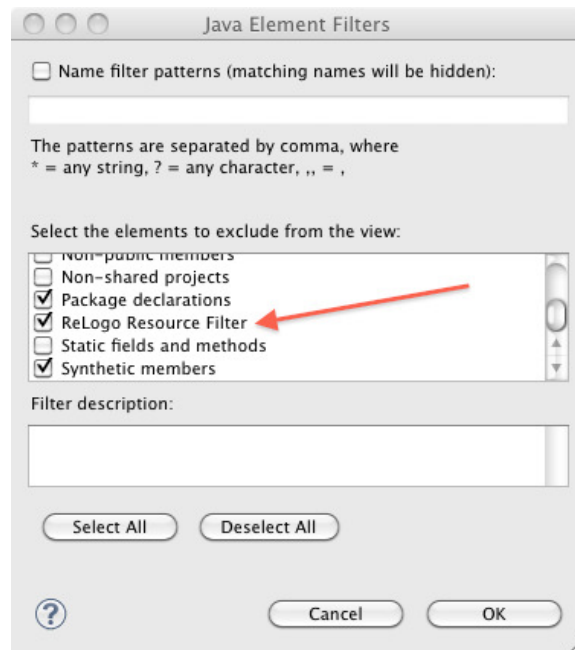


FIGURE 4. Filter Window.

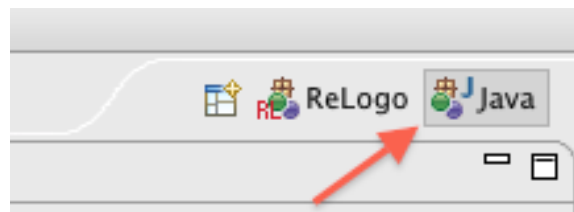


FIGURE 5. Selecting the Java Perspective.

(1) Step 3: GridCell cell in gridCells

Decision Check count

(1) Step 3: cell.size() > maxCount

Task Point with most humans

(1) pointWithMostHumans = cell.getPoint()

(2) maxCount = cell.size()

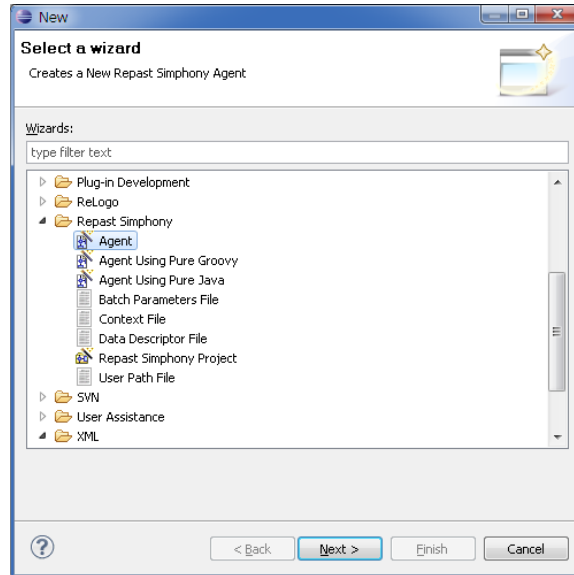


FIGURE 6. New Agent Wizard Step 1

Task Move and Infect

- (1) `int x = pointWithMostHumans.getX()`
- (2) `int y = pointWithMostHumans.getY()`
- (3) `grid.moveTo(this,x,y)`
- (4) `moved = true`
- (5) `infect()`

Finally, we need to define the Infect behavior for the Zombie agent. The completed flowchart for the agent is shown in (Fig. 11).

Behavior Infect

- (1) Step 7: Type in a Compiled Name : `infect`

Task Find Humans

- (1) `Grid grid = FindGrid("FlowZombies/grid")`
- (2) `GridPoint pt = grid.getLocation(this)`
- (3) `List humans = new ArrayList()`
- (4) `Iterable objects = grid.getObjectsAt(pt.getX(), pt.getY())`

Loop over objects

- (1) Step 3: `objects.hasNext()`

Task Get Object

- (1) `Object o = objects.next()`

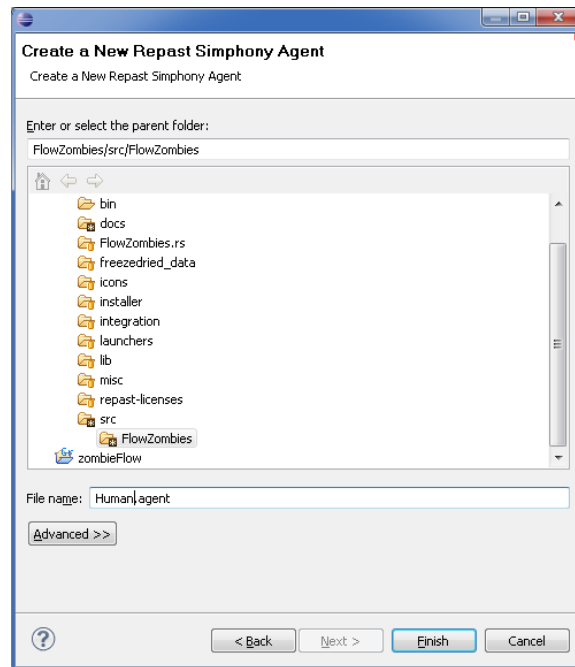


FIGURE 7. New Agent Wizard Step 2

Decision is Human?

(1) Step 3: o instanceof Human

Task save human

(1) humans.add(o)

Decision Found Humans?

(1) Step 3: humans.size() > 0

Task Braaaaaains!

(1) int index = RandomHelper.nextIntFromTo(0, humans.size() - 1)

(2) Object human = humans.get(index)

(3) Context context = RemoveAgentFromContext("FlowZombies", human)

(4) Object zombie = CreateAgents("FlowZombies", "FlowZombies.Zombie", 1)

(5) MoveAgent("FlowZombies/grid", zombie, pt.getX(), pt.getY())



FIGURE 8. Zombie Agent Step 1



FIGURE 9. Zombie Agent Step 2

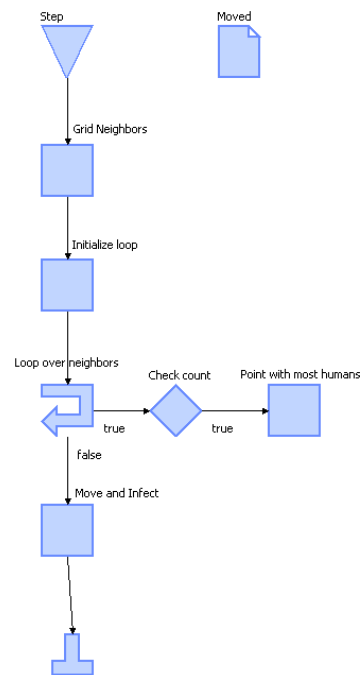


FIGURE 10. Zombie Agent Step 3

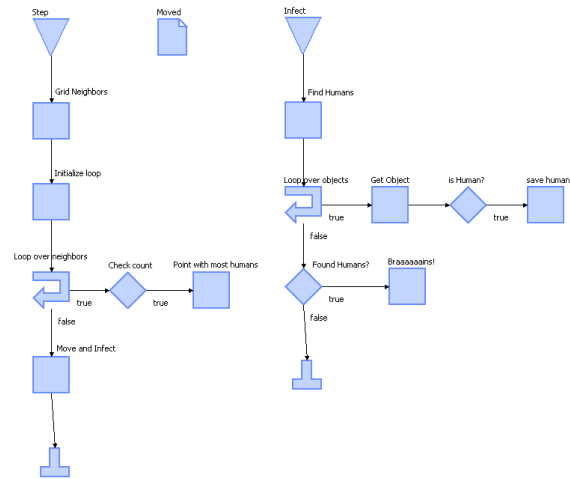


FIGURE 11. Zombie Agent Completed

Let's now turn to the code for the Humans. Select Human.agent in Eclipse's editor pane. The basic behavior for a Human is to react when a Zombie comes within its local neighborhood by running away from the area with the most Zombies. Additionally, Humans have a certain amount of energy that is expended in running away. If this energy is 0 or less then a Human is unable to run. We begin by creating properties of **Energy** and **Starting Energy**. **Energy** will be used to track the current amount of energy a Human has. **Starting Energy** will be used to set the energy level back to its starting level after a Human has had a rest. The main behavior of a Human is implemented in its **run** Behavior. The starting point for the Human flowchart is shown in (Fig. 12) Create the properties for Energy and Starting Energy and create the Run Behavior.

Property Energy

- (1) Step 2: Energy
- (2) Step 3: energy
- (3) Step 5: int
- (4) Step 6: 10

Property Start Energy

- (1) Step 2: Start Energy
- (2) Step 3: startEnergy
- (3) Step 5: int
- (4) Step 6: 10

Behavior Run

- (1) Step 4b: Type in a Query for the Trigger Condition : within_vn 1
- (2) Step 4c: Type in Kind of Agents to Watch : FlowZombies.Zombie
- (3) Step 4d: Type in a Comma Separated List of Target Agent Fields To Watch : moved
- (4) Step 4f: Type in a Delay Kind Before the Behavior Triggers : Immediate
- (5) Step 4g: Type in a Delay Time Before the Behavior Triggers : 0

Next, we will provide the mechanism for the Run Behavior. Fig. 12 shows the completed Human flowchart. Enter the details for each of the following flowchart blocks:

Task Grid Neighbors

- (1) Grid grid = FindGrid("FlowZombies/grid")
- (2) GridPoint pt = grid.getLocation(this)
- (3) GridCellNgh nghCreator = new GridCellNgh(grid, pt, Zombie.class,1,1)
- (4) List gridCells = nghCreator.getNeighborhood(true)
- (5) SimUtilities.shuffle(gridCells, RandomHelper.getUniform())

Task Initialize loop

- (1) GridPoint pointWithLeastZombies = null
- (2) int minCount = Integer.MAX_VALUE

Loop over neighbors

- (1) Step 3: GridCell cell in gridCells

Decision check count

- (1) `cell.size() < minCount`

Task Point with least zombies

- (1) `pointWithLeastZombies = cell.getPoint()`
- (2) `minCount = cell.size()`

Decision Check energy

- (1) Step 3: `energy > 0`

Task Move

- (1) `int x = pointWithLeastZombies.getX()`
- (2) `int y = pointWithLeastZombies.getY()`
- (3) `grid.moveTo(this,x,y)`
- (4) `energy--`

Task Reset Energy

- (1) `energy = startingEnergy`



FIGURE 12. Human Agent Start

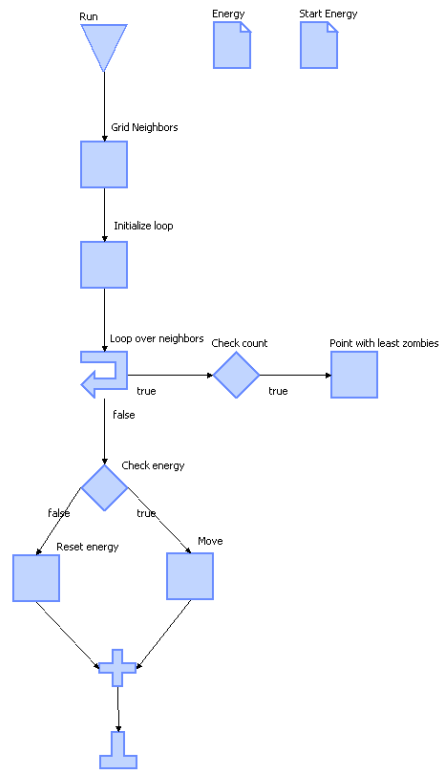


FIGURE 13. Human Agent Completed

This looks much like the Zombie code. A `GridCellNgh` is used to find the Zombies in the neighboring grid cells. It then determines which of these cells has the least Zombies and attempts to move towards that. Note that `moveTo` is only called if the energy level is greater than 0. If energy does equal 0, the Human doesn't move and energy is set back to its starting level. Unlike the Zombie code we are not going to schedule the `run()` method for execution. Rather we are going to setup a *watcher* that will trigger this `run()` method whenever a Zombie moves into a Human's neighborhood. The watcher will trigger whenever this field has been accessed.

This `Watch` will watch for any changes to a "moved" property in the Zombies agent. What this means is whenever any Zombie moves and their moved variable is updated, then this `Watch` will be checked for each Human. If the query returns true for that particular Human then `run` will be called immediately on that Human. Our query will return true when the Zombie that moved is within the Moore neighborhood (8 surrounding grid cells) of the Human whose `Watch` is currently being evaluated.

That completes this part of the code for Zombies and Humans. Now we need to turn to initializing the simulation which is achieved with a special type of agent called the "ModelInitializer." The ModelInitializer agent is used to hold actions done during the simulation initialization, such as creating agents. Create the ModelInitializer agent in the same way as was done with the Human and Zombie agents through the new agent wizard. It is important that the agent name is exactly "ModelInitializer" as Repast will look for this specifically named agent.

First, create a behavior in the ModelInitializer that will start the initialization process. This is a behavior step that will occur only one time at the very beginning of the simulation.

Behavior Initialize Model

- (1) Step 2: Initialize Model
- (2) Step 4f: Immediate
- (3) Step 7: initializeModel

Note that in Step 7 it is important that the property value is exactly "initializeModel" as this is a special property name used by the ModelInitializer agent. Create an End step and connect as shown in Fig. 14. This initialize behavior and end step are the minimum required steps in the ModelInitializer to create a runnable simulation. Without these steps, an error will be produced when the model is initialized. Of course, the model initializer behavior is still empty so we need to add steps to create the Human and Zombie models.

We will create a specified number of Zombies and Humans by looping through some creation code a specified number of times. We then add the new Zombies and Humans to context. In adding them to the context we automatically add them to any projections associated with that context. The completed Model Initializer agent is shown in Fig. 15.

Property Human Count

- (1) Step 2: Human Count
- (2) Step 3: humanCount
- (3) Step 5: int
- (4) Step 6: 200

Property Zombie Count

- (1) Step 2: Zombie Count
- (2) Step 3: zombieCount
- (3) Step 5: int
- (4) Step 6: 5

Loop Count Humans

- (1) Step 3: i in 1..humanCount

Task Create Humans

- (1) Object agent = CreateAgents("FlowZombies", "FlowZombies.Human",1)
- (2) Human human = (Human)agent
- (3) human.energy = RandomHelper.nextIntFromTo(4, 10)
- (4) human.startEnergy = human.energy

Task Create Zombies

- (1) CreateAgents("FlowZombies", "FlowZombies.Zombie",zombieCount)

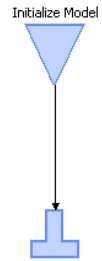


FIGURE 14. Model Initializer Start

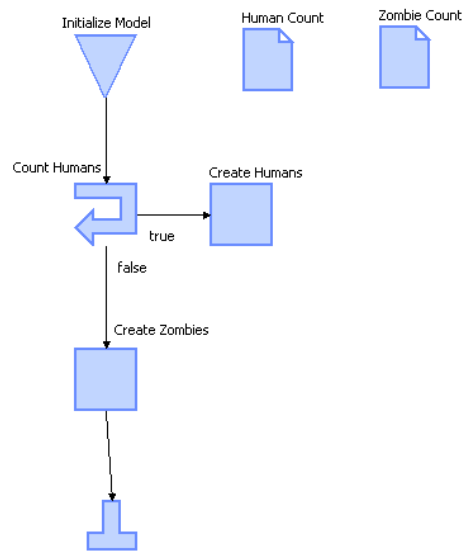


FIGURE 15. Model Initializer Completed

Before we run this model, we need to update the metadata that the Repast Symphony runtime uses to help create displays and other runtime components. Open the FlowZombies.rs folder under the project folder, and double click on the `context.xml` file.

That should bring up the xml editor for editing the `context.xml` file. The `context.xml` file describes the context hierarchy for your model. The context hierarchy is composed of the contexts your model uses and the projections associated with them.

We need to add a Grid projection using the editor. To do that,

- (1) Right click on the context element and select “Add Child”, then choose “projection”.
- (2) Expand the projection element by clicking on the triangle next to the newly added projection.
- (3) Choose “grid” from the drop down box to set the value of the type attribute
- (4) Type “grid” (no quotes) for the id of the projection.
- (5) Right click on the new projection element and select “Add Child”, then choose “attribute”.
- (6) Expand the attribute and set its id, type and value properties to width, int, and 50 respectively.
- (7) Add another attribute to the projection and set its id, type and value properties to height, int, and 50.
- (8) Add another attribute to the projection and set its id, type, and value properties to border rule, string, periodic.
- (9) Add another attribute to the projection and set its id, type, and value properties to allows multi, boolean and true.

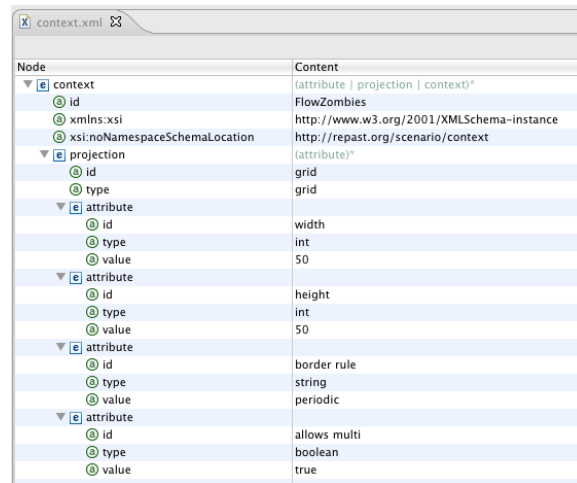
The `context.xml` editor should now look like Fig. 16.

Now its time to launch the model. When we created our project using the Repast Symphony Project Wizard, it automatically created Eclipse launchers for us, and we can use those to launch the model. If you click on the small downward facing triangle next to the Eclipse launcher button (fig. 17), you’ll see the various available launchers. Click on “FlowZombies Model”.

When the model window appears, we can observe the Scenario Tree on the left hand side of the window. The Scenario Tree allows us to customize things like model initialization, data logging, and displays. We first need to specify how the model is initialized and we will do so with the `ModelInitializer` agent we created. In the Scenario Tree right click on the Data Loaders and select “Set Data Loader.” A list appears with the various types of data loading options available. Select “Context.xml file and `ModelInitializer`” and then click Next and Finish.

Next, we will now create a simple display.

- (1) Right click on Displays in the Scenario Tree and click “Add Display”
- (2) In the Display configuration dialog, type Grid Display for name. Leave 2D as the type.



Node	Content
context	(attribute projection context)*
id	FlowZombies
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation	http://repast.org/scenario/context
projection	(attribute)*
id	grid
type	grid
attribute	
id	width
type	int
value	50
attribute	
id	height
type	int
value	50
attribute	
id	border rule
type	string
value	periodic
attribute	
id	allows multi
type	boolean
value	true

FIGURE 16. context.xml location

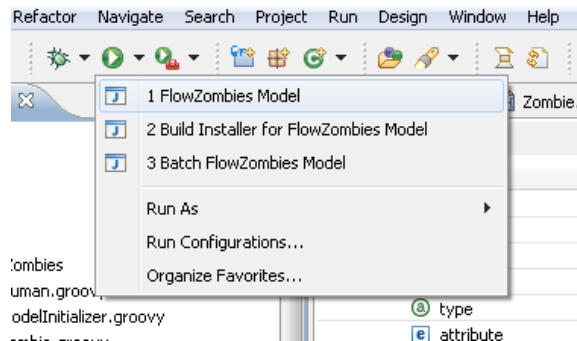


FIGURE 17. FlowZombies Model Launcher

- (3) Select our “grid” projection as the one we want to display. Click on space in the “Projection and Value Layers” section and then click the green arrow. The projections on the right are the ones will be displaying and those on the left are all the possible projections to display. The dialog should now look like fig. 18
- (4) Click Next.
- (5) Select the Human and Zombie agents as the types we want to display. Do this by selecting each in the left and then clicking the right pointing arrow to move them

to right. If Zombie is not at the top of the list on the left use the up and down arrows to move it to the top. The dialog should now look like fig. 19

- (6) Click Next.
- (7) In this panel, we can configure what we want the Zombies and Humans to look like. This can be done programmatically by specifying a class that implements `repast.simphony.visualizationOpenGL2D.StyleOpenGL2D` or via a wizard. At this point you have two options, you can use the icons from the completed FlowZombies model to represent your agents or use a simpler style. We present the simpler style first.
- (8) Simple Style
 - (a) Click the button to the right of the style class combo box (fig. 20).
 - (b) In the 2D Shape editor, change the Icon Shape to a square using the combo box and change the color to red by clicking the button with the blue square, and choosing a red color from the icon color dialog. Click OK on the icon color box, then OK on the 2D Shape Editor.
 - (c) Repeat the previous step for the Human. Click on Human in the list of Agents. Then click the icon editor button as before. Leave the default as is, and click OK.
- (9) Icon Style
 - (a) Click the button to right of the style class combo box (fig. 20).
 - (b) In the 2D Shape editor, click on the “Select Icon File” button. We want to use the zombie.png icon that comes with the jzombies demo model. Navigate to where the demo models are installed and click on zombies.png in the `jzombies/icon` directory. (If you can’t find zombies.png, feel free to style the Zombie as a circle or whatever, using the 2D Shape editor).
 - (c) Click OK when you have finished.
 - (d) Repeat the previous step for the Human. Click on Human in the list of Agents. Then click the icon editor button as before. Click the “Select Icon File” button and navigate to the `jzombies/icon` directory in the demo model. Choose the person.png icon.
 - (e) Click OK when you have finished with the 2D Shape editor.
- (10) Click Next
- (11) Click Next
- (12) Click Finish

You should now see “Grid Display” under the Display node in the Scenario Tree. Save your new scenario info (the new Data Loader and Display) by clicking the “Save” button (fig. 21) on the Repast Symphony runtime toolbar.

We can now run our model. Click the Initialize button (fig. 22) to initialize the model and bring up the display. If the display is not centered, you can center it by clicking on the display “home” button to reset the view. The mouse wheel will zoom the display in and out as will holding down the shift key and right mouse button and moving the mouse

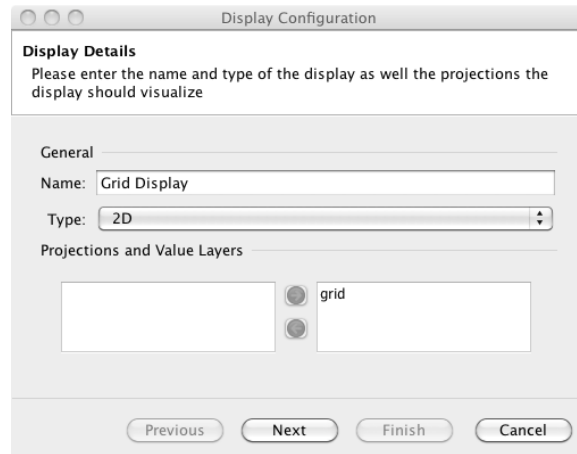


FIGURE 18. Configuring the Display

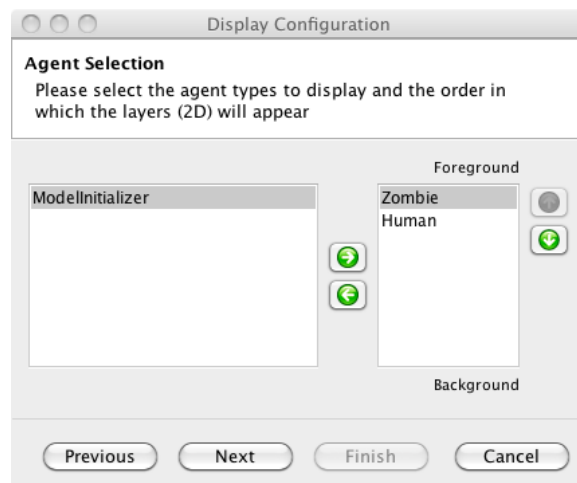


FIGURE 19. Configuring the Display 2

up and down. You can run the simulation by clicking the Run button, step through each timestep with the Step button, stop the simulation with the Stop button, and reset it for another run with the Reset button (fig. 22). When the simulation has been stopped, you must reset it with the Reset button in order to do any more runs.

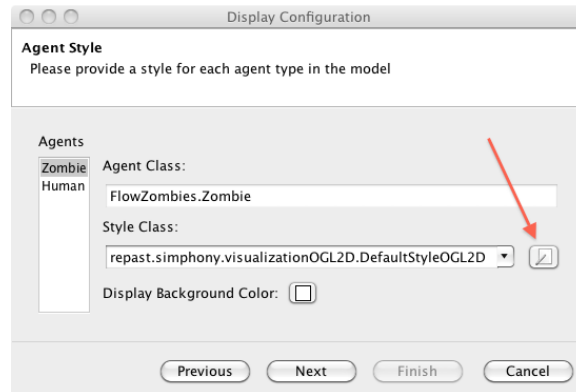


FIGURE 20. Configuring the Display 3

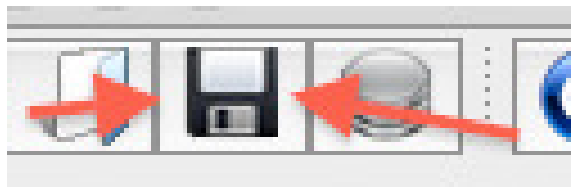


FIGURE 21. Save Scenario Button

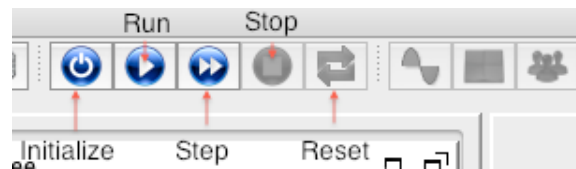


FIGURE 22. Repast Simphony Simulation Buttons

Once the Repast Simphony runtime has come up, initialize and run the simulation. You should now see humans becoming zombies. If you don't, then check your code for mistakes.

1.2. Model Distribution. Repast models can be distributed to model users via the installation builder. This feature packs up your model and all of the software you need to run it, except for a properly configured Java Runtime Environment, into a single Java archive

("JAR") file that can be given to model users. The resulting installer can be executed on any system with Java version 1.8 or later. Users simply copy the installer file onto their Windows, Mac OS, or Linux computers and then start the installer by double clicking on the file. Once the installer is started it will show an installation wizard that will prompt the user for the information needed to install the model. If desired, the installer can also be run in a command line mode.

Building an installer for a model is straightforward. Simply choose the "Build Installer for <Your Model Name Here> Model" and provide a location and name for the installer file. The installer file's default name is "setup.jar," which is suitable for most purposes. The install builder will then package and compress your model and the supporting Repast software. The resulting installer files are about 70 MB plus the size of the model code and data. 75 MB to 80 MB is a common total size.

The Repast install builder uses the [IzPack system](#). More information on installer customization and use, including command line activation, can be found on the [IzPack web site](#).